CSE3201 Project

Advertising Panel

Submitted by:
Hassan Chehaitli (209003898), **Aysar Khalid** (209728866)

Submitted on:
Tuesday Dec 20th, 2011.

INTRODUCTION

In this project we will design, implement, and an advertising panel. The essence of the ad panel is that it will be LCD-powered with the ability to apply live textual effects on the input. The input is interpreted from the computer onto the LCD. Secondary inputs are used to determine the type of effects to apply onto the text input; this is done using SW switches. The output will be displayed onto the LCD and using the input from the KEY buttons we will apply the corresponding effects onto the text input. The text with the effects applied will then be displayed onto the LCD. Effects applied are: text scrolling (horizontal sliding) and text flashing.

Specifically the two text effects or user options are wired as follows. The first option (scrolling text effect) is available when SW [10] is turned on. When SW [10] is turned on we can either make the text scroll (traverse horizontally) or we can make it stable (simply display text normally), text scroll when SW [0] is on, and stable when SW [0] is off. The second option (text flashing) is when SW [10] is off; we can make the text blink by repetitively turning on and off SW [1].

OBJECTIVE

The purpose of this experiment is to get used and acquainted with the design of combinational circuit and how it interacts with the DE2 board. Also how to introduce different modules and how they are integrated with each other in a cohesive manner to output the desired result. In this case an advertising panel.

APPARATUS

- Computer
- DE2 board
- USB cable
- DE2 board adapter

DESIGN

In this section we are going to discuss different modules our project is made of, and what they do and how they interact together to deliver the required result.

The main module is lcdlab2.v; this main module is made of two modules (reset_delay.v and lcd_controller.v) and some code written in the main module to synchronize the interaction between the two modules for processing. First we will discuss the inner modules and what they do, and then we will link all the modules with the written code in order to get the required results.

reset_delay.v:

```
module  Reset_Delay( input iCLK, output reg oRESET);
reg     [19:0]  Cont;

always@(posedge iCLK)
begin
        if(Cont!=20'hFFFFF)
        begin
                Cont    <=      Cont + 1'b1;
                oRESET  <=      1'b0;
        end
        else
        oRESET  <=      1'b1;
end

endmodule
```

This module plays a role as a timer. This timer on regular basis checks if there is a command that is ready to be sent to LCD to be executed; this module plays an important role to prevent flooding the LCD with commands and thus prevents it from crashing.

It is made of one input and one output. In our case the input (iCLK) is the 50 MHz clock. And the output is oRESET which is generated every constant period of time to make the lcd_controller module to check if there is any command that needs to be executed in order to send to LCD.

Whenever the counter reaches the value of “FFFFF” the signal is generated and is sent to lcd_controller.

The clock is executed on posedge, thus the counter is incremented by 1 every 1 cycle of the clock. 1 cycle = $T_{clock} = 1 / frequency_{(50\ Mhz)}$.

The equivalent time for this counter is FFFFF X T_{clock} = 1048575 * 1/50,000,000 = 0.0209 = 2.09 ms. Thus every 2.09 ms a signal will be generated in order to make the lcd_controller to check if there is any command need to be send and executed on the LCD panel.

Lcd_controller.v

```
module LCD_Controller (
//      Host Side
input [7:0] iDATA,
input iRS,
input iStart,
output reg oDone,
input iCLK,iRST_N,
//      LCD Interface
output [7:0] LCD_DATA,
output LCD_RW,
output reg LCD_EN,
output LCD_RS   );

parameter       CLK_Divide      =       16;

//      Internal Register
reg             [4:0]   Cont;
reg             [1:0]   ST;
reg             preStart,mStart;

////////////////////////////////////////////////
//      Only write to LCD, bypass iRS to LCD_RS
assign  LCD_DATA        =       iDATA;
assign  LCD_RW          =       1'b0;
assign  LCD_RS          =       iRS;
////////////////////////////////////////////////
```

This module plays an important role in controlling LCD, DATA or COMMANDS are passed to this module and this module waits for the right time to pass data or commands to LCD to be printed or performed, it plays a role in preventing data or command flooding to LCD.

We have input “iDATA”, this input holds data or command to be sent to LCD.

Input “iStart”, if iStart is 1 that means there is data or a command that needs to be passed or executed, if iStart is 0 that either means there is nothing to pass to LCD or there is a previous data or command passed to LCD and still needs time to be executed.

Input “iRST_N” this input comes from the reset_delay module, it comes every 2.09 ms; the function of reset_delay has been explained.

Input “iRS”, this input indicates whether the bits in “iDATA” represents data or command, if iRS is 0 then the bits represent command else represent data.

Outputs “LCD_DATA”,”LCD_RW”,”LCD_EN”,”LCD_RS”. All these outputs are sent to the LCD, LCD behaves according to the information in these variables.

Output “oDone”. This output tell the main module(lcdlab2) that the command or data has been sent to LCD, and the main module should start a timer and wait till the execution of the command or data is printed on the LCD has finished.

assign LCD_DATA=iDATA; ---- 1

assign LCD_RW=1'b0; ---- 2

assign LCD_RS=iRS; ---- 3

lines 1,2 and 3 are always set to the values on the right hand of their equal statements.

LCD_RW is always 0 because whether it is data or a command LCD_RW should be 0.

These instructions are not executed until LCD_EN is set to 1.

```
always@(posedge iCLK or negedge iRST_N)
begin
        if(!iRST_N)
        begin
                oDone   <=      1'b0;
                LCD_EN  <=      1'b0;
                preStart<=      1'b0;
                mStart  <=      1'b0;
                Cont    <=      0;
                ST              <=      0;
        end
        else
        begin
                //////  Input Start Detect ///////
                preStart<=      iStart;
                if({preStart,iStart}==2'b01)
                begin
                        mStart  <=      1'b1;
                        oDone   <=      1'b0;
                end
                ////////////////////////////////////
```

The code above plays an important role to whether LCD_EN should be set 1, when it is set to 1 data or commands are passed to LCD.

If iRST_N(comes from reset_delay.v) is 0, then that means the time for sending the new command is not over yet. iRST_N is important for not overloading LCD with commands, every time iRST_N is set to 1 that means 2.09 ms has passed.

Every 2.09 ms (iRST_N = 1) we check if we have a new command or data to be passed to LCD. Whenever 2.09 ms passes, it goes in the “else” statement block, it checks if new command is available.

In order to prevent multiple commands to be sent to LCD, we check the previous state of “iStart” (preStart) and check if the value has changed from 0 to 1 only then a new command is available.

If the value changes from 0 to 1 then we will prepare to send the new command to LCD by setting “mStart” to 1.

```
                if(mStart)
                begin
                        case(ST)
                        0:      ST      <=      1;      //      Wait Setup
                        1:      begin
                                        LCD_EN  <=      1'b1;
                                        ST              <=      2;
                                end
                        2:      begin
                                        if(Cont<CLK_Divide)
                                        Cont    <=      Cont + 1'b1;
                                        else
                                        ST              <=      3;
                                end
                        3:      begin
                                        LCD_EN  <=      1'b0;
                                        mStart  <=      1'b0;
                                        oDone   <=      1'b1;
                                        Cont    <=      0;
                                        ST              <=      0;
                                end
                        endcase
                end
        end
end

endmodule
```

The code above is for sending the command to LCD, when there is a new command. In case “1” LCD_N is set to 1 so that the LCD can start processing the command or data saved in LCD_DATA, and behaviour is determined by the values of LCD_RW and LCD_RS.

After LCD_EN has been set to 1(in state2), a small waiting time is imposed(in state 3) in order to make sure the LCD gets the information from its input values, after that LCD_EN is disabled (since data or command has already been passed) and oDone has been set 1(in state 4), and control is passed to the main module(lcdlab2.v) so that it stops new data or commands from being sent to LCD till command execution time is over, we are going to see in details what happens in lcdlab2 module.

lcdlab2.v

this is the main module it has 2 always loops. 1 loop is to check if new command or data should be passed to lcd_controller module and wait till the command is finished execution. And the other always loop is to choose the required command or data to be sent to lcd_controller.

1st always loop

```
always@(posedge CLOCK_50 or negedge DLY_RST or posedge lcd_start)
begin
        if(!DLY_RST || lcd_start)
        begin
                LUT_INDEX       <=      0;
                mLCD_ST         <=      0;
                mDLY            <=      0;
                mLCD_Start      <=      0;
                mLCD_DATA       <=      0;
                mLCD_RS         <=      0;
        end
        else
```

This loop is invoked for 2 reasons, 1st with CLOCK_50 the reason for this is that when there is a command or data being process the loop should be keeping track of the elapsed time. As for DLY_RST we need so that we can check every 2.09 ms if we have a new command or data to be sent to lcd_controller.

The if statement means be in standby mode till DLY_RST signal is given, when it is given check for commands and data to be executed and process them and wait for them till they are done and repeat steps again, As for lcd_start it is just used once at the beginning waiting for the LCD to get ready.

```
        else
                begin
                case(mLCD_ST)
                0:      begin
                                mLCD_DATA       <=      LUT_DATA[7:0];
                                mLCD_RS         <=      LUT_DATA[8];
                                mLCD_Start      <=      1;
                                mLCD_ST         <=      1;
                        end
                1:      begin
                                if(mLCD_Done)
                                begin
                                        mLCD_Start      <=      0;
                                        mLCD_ST         <=      2;
                                end
                        end
                2:      begin
                                if(mDLY<18'h3FFFE)
                                mDLY    <=      mDLY + 1'b1;
                                else
                                begin
                                        mDLY    <=      0;
                                        mLCD_ST <=      3;
                                end
                        end
                3:      begin
                                if (LUT_INDEX<LUT_SIZE) begin
                                        mLCD_ST <= 0;
                                        LUT_INDEX <= (LUT_INDEX + 1'b1);
                                end
                        end
                endcase
        end
end
```

When the DLY_RST signal is given processing of new commands and data are given, in “case 0” the next available information is stored in LUT_DATA, and the type of the information whether it is command or data is stored in LUT_DATA[8]. mLCD_Start is set to 1 to indicate lcd_controller that it can process this information on the next DLY_RST signal as explained before.

Then we go to “case 1”, in this case we wait till the lcd_controller indicates that the information has been sent to LCD and that we can start the timer for information processing.

In “case 2” the timer is started and when it is over it is going to going to next case to grab the next information on the list to be processed, the timer waits for 262142/50000000 = 0.524 ms.

In “case 3” the pointer of the information is incremented in order to get the next information need to be processed and thus the 2nd always loop is invoked.

2nd always loop

```
always @(posedge CLOCK_50)
begin


if(SW[10] == 1)
        begin
                var = SW[0];
                varl = 2;
        end
        else
        begin
                varl = SW[1];
                var = 2;
        end
```

This section decides which effect to apply to text according to the switches turned off or on.

If SW [10] is turned on, and SW [0] is off it show a stable text, if SW [0] is on it shows a scrolling text.

If SW [10] is off, and SW [1] is turned on and off the text will flash on the LCD.

```
if(var == 1)
        begin
        case(LUT_INDEX)
        0:      LUT_DATA        <=      9'h038;
        1:      LUT_DATA        <=      9'h01C;
        2:      LUT_DATA        <=      9'h011;
        3:      LUT_DATA        <=      9'h016;
        4:      LUT_DATA        <=      9'h080;
        //      Line 1
        LINE1+0:                LUT_DATA        <=      {1'b1,out0};
        LINE1+1:                LUT_DATA        <=      {1'b1,out1};
        LINE1+2:                LUT_DATA        <=      {1'b1,out2};
        LINE1+3:                LUT_DATA        <=      {1'b1,out3};
        LINE1+4:                LUT_DATA        <=      {1'b1,out4};
        LINE1+5:                LUT_DATA        <=      {1'b1,out5};
        LINE1+6:                LUT_DATA        <=      {1'b1,out6};
        LINE1+7:                LUT_DATA        <=      {1'b1,out7};
        LINE1+8:                LUT_DATA        <=      {1'b1,out8};
        LINE1+9:                LUT_DATA        <=      {1'b1,out9};
        LINE1+10:       LUT_DATA        <=      {1'b1,out10};
        LINE1+11:       LUT_DATA        <=      {1'b1,out11};
        LINE1+12:       LUT_DATA        <=      {1'b1,out12};
        LINE1+13:       LUT_DATA        <=      {1'b1,out13};
        LINE1+14:       LUT_DATA        <=      {1'b1,out14};
        LINE1+15:       LUT_DATA        <=      {1'b1,out15};

        default:
                LUT_DATA        <=      9'h120 ;
        endcase
        end
```

This is the command and data list that gives the scroll effect; other effects are similar in construction with sum differences.

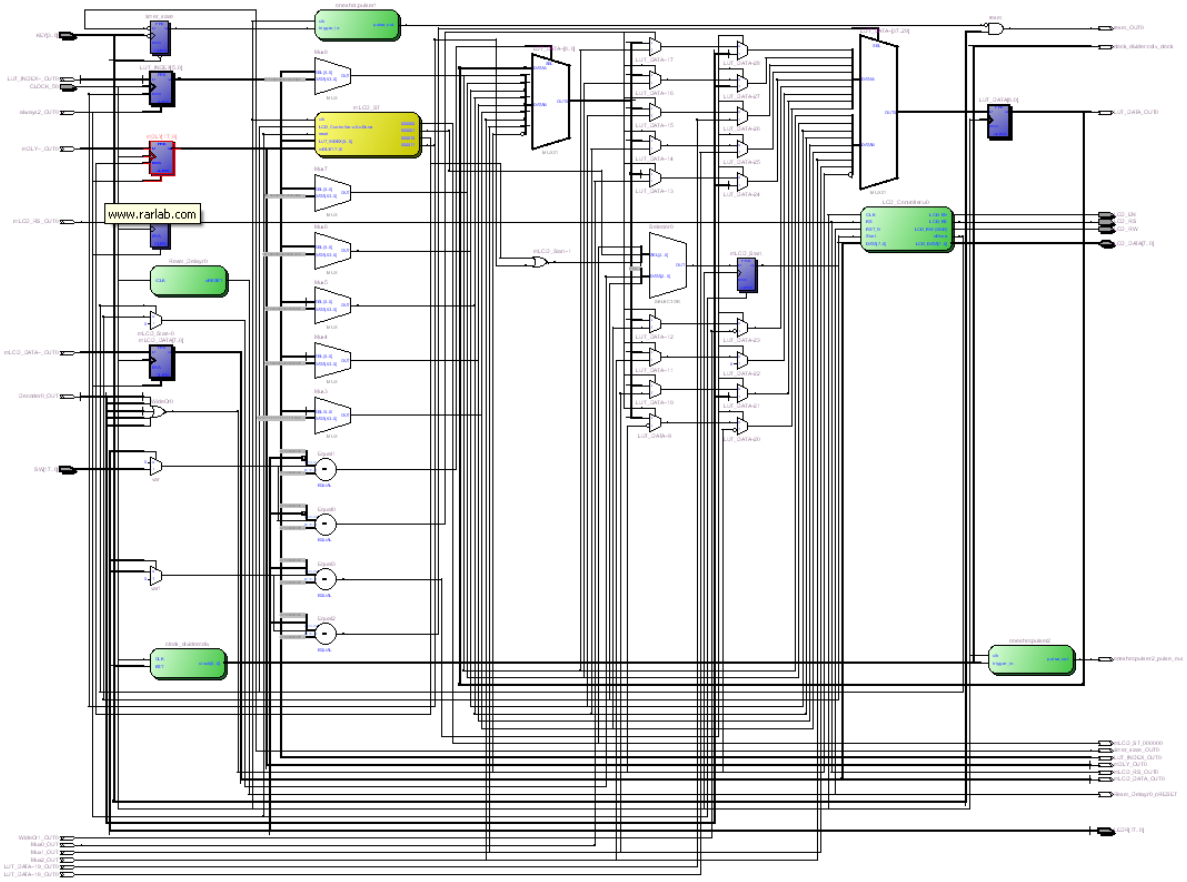
From case 0→4 these are commands that are sent to LCD

Command at “0” 9’h038 represent function set, with the following options, with data length DL = 8 bits, number of lines N = 2, and font type (F:5X11dots)

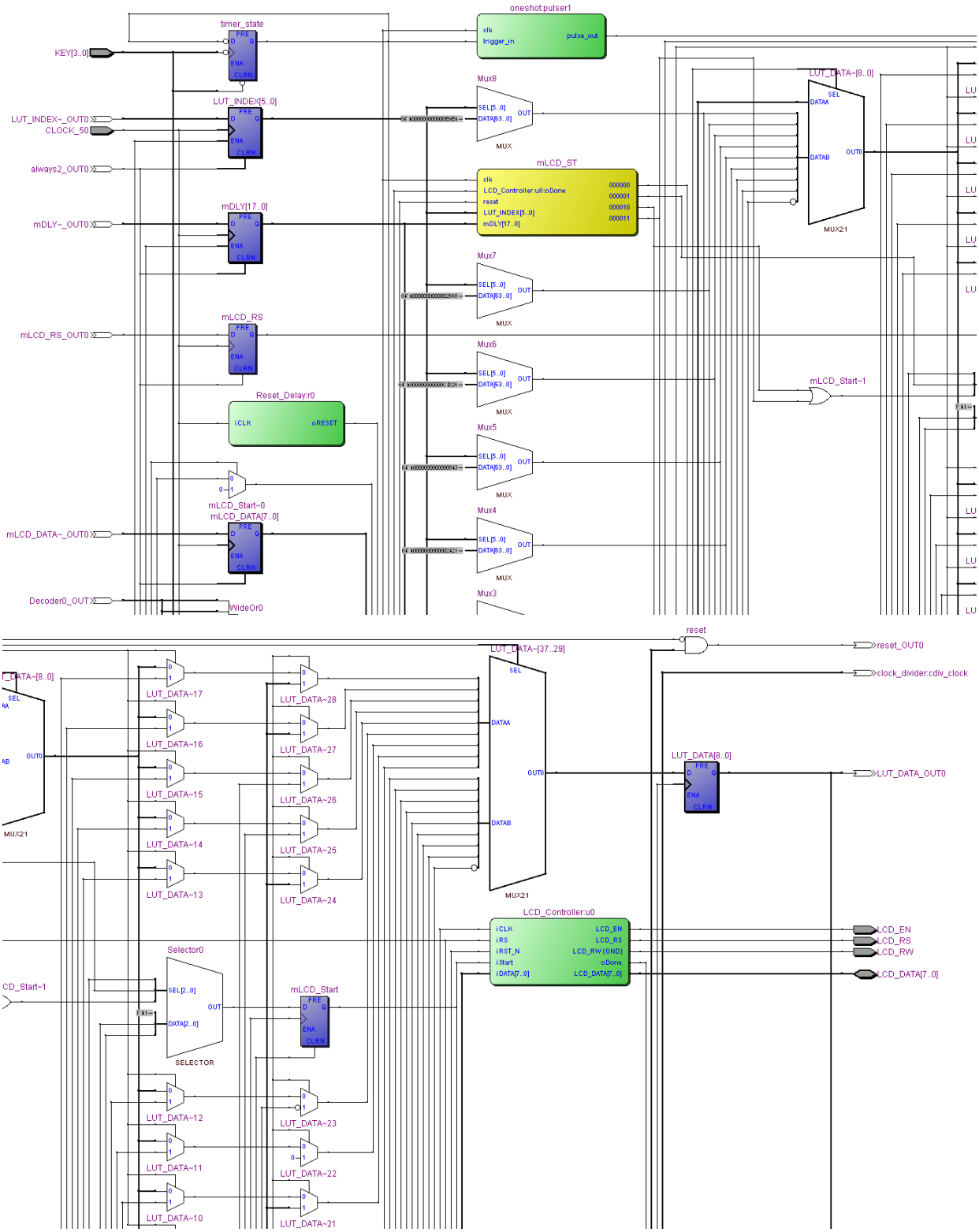
Command at “1” 9’h01c it is to set cursor moving and display shift and the direction without changing the DDRAM data (it gives the scroll effect). The commands from 2→ are for clearing the screen and get the cursor back home every time new printing needs to be done. As for the case of LINE1+i, this represents what’s to be printed at the specified line and “i” represents the column it is going to be printed at. As for the rest of the effects it is similar to the above code with the change of command “1” from 9’h01c to 9’h00c to prevent scrolling.

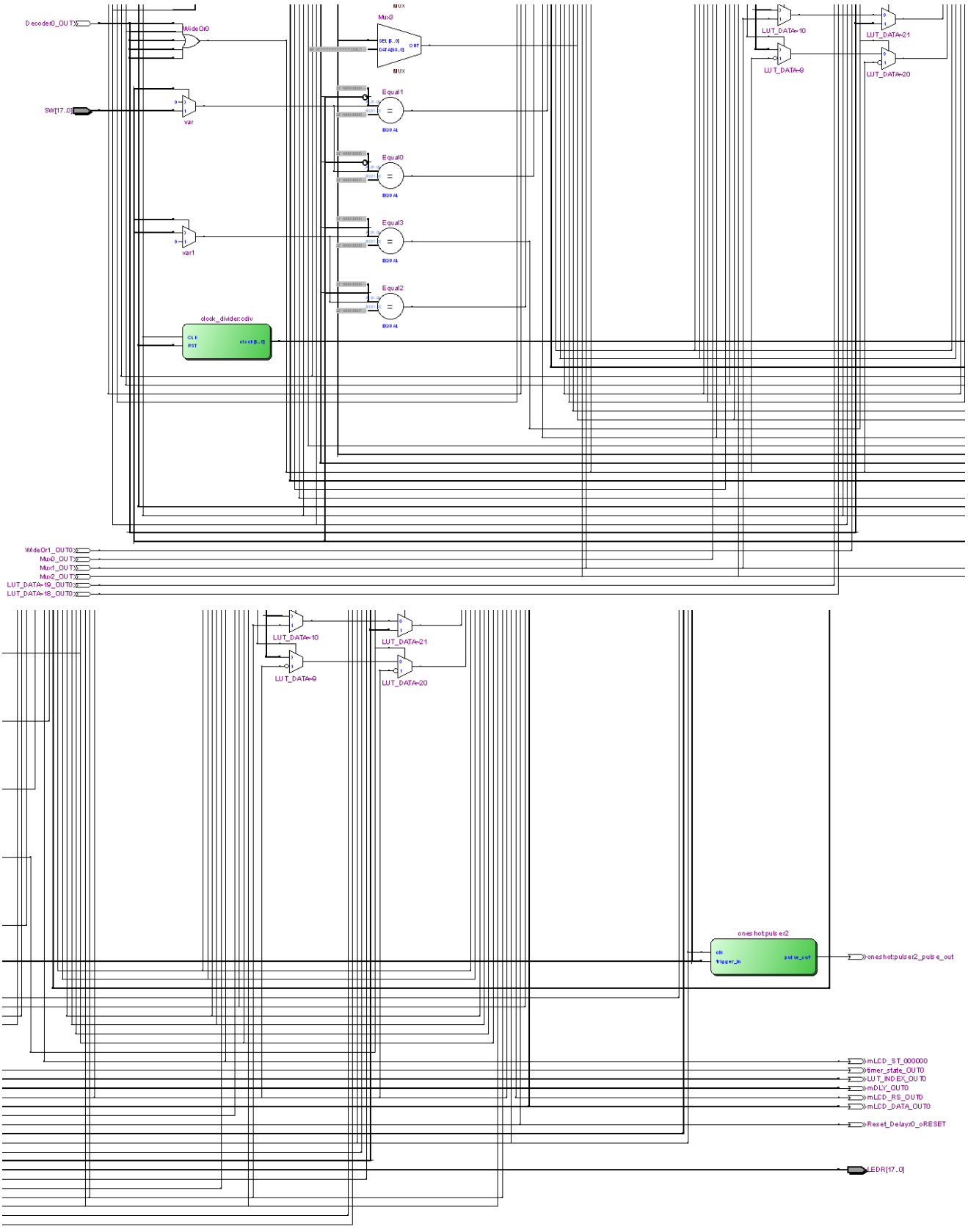
DIAGRAMS

The following is the entire RTL view of the project.

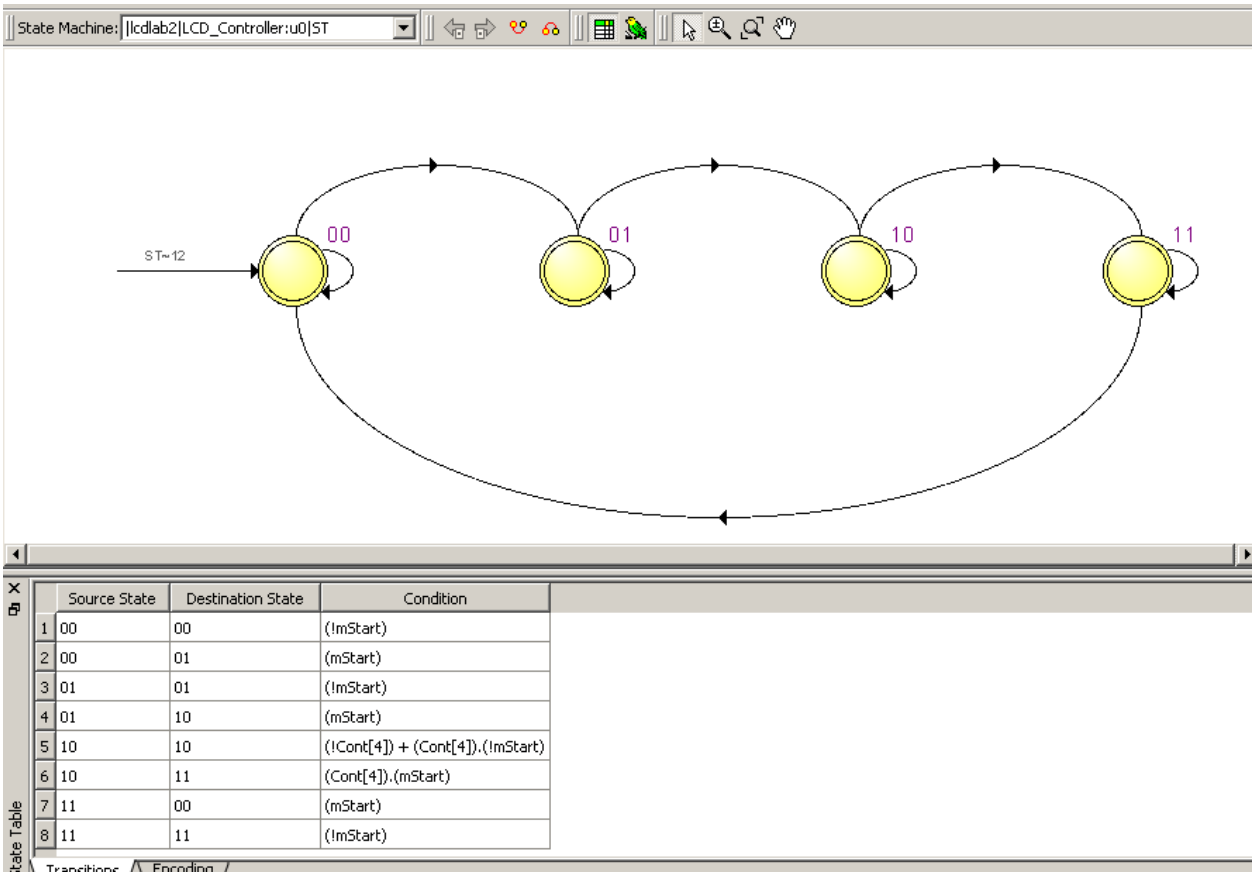


The following is a detailed view of the RTL, starting from top-left and ending at bottom-right




Decoder0_OUT[7..0]
WideOr0
SW[17..0]
var
var1
MUX
Mux3
SEL[5..0]
DATA[63..0]
OUT
Equal1
Equal0
Equal3
Equal2
clock_divider:cdiv
CLK
RST
clock[5..0]
LUT_DATA~10
LUT_DATA~9
LUT_DATA~21
LUT_DATA~20
WideOr1_OUT[0..0]
Mux0_OUT[3..0]
Mux1_OUT[3..0]
Mux2_OUT[3..0]
LUT_DATA~19_OUT[0..0]
LUT_DATA~18_OUT[0..0]
oneshotpulser2
clk
trigger_in
pulse_out
oneshotpulser2_pulse_out
mLCD_ST_000000
timer_state_OUT[0..0]
LUT_INDEX_OUT[0..0]
mDLY_OUT[0..0]
mLCD_RS_OUT[0..0]
mLCD_DATA_OUT[0..0]
Reset_Delay:r0_oRESET
LEDR[17..0]

The following is the state machine view of the main module.



	Source State	Destination State	Condition
1	00	00	(!mStart)
2	00	01	(mStart)
3	01	01	(!mStart)
4	01	10	(mStart)
5	10	10	(!Cont[4]) + (Cont[4]).(!mStart)
6	10	11	(Cont[4]).(mStart)
7	11	00	(mStart)
8	11	11	(!mStart)

State Table
Transitions
Encoding

DIFFICULTIES

The difficulty that faced us is the difference between the Verilog compilers. As we wrote the code at home and it compiled with no errors but when compiled on the QUARAS II it gave errors. Some difficulties was observed with “always” statement as it used to execute for no reason when there is no change in key value so we used clock instead. Till now we couldn’t know what’s wrong with it. We used if statements instead of cases in our program and it was interacting in a weird way, so we had to change back to the case programming syntax.

CONCLUSION

In this lab we learned how to use advanced DE2 board settings and how the Verilog language interacts with the board through various input and output pin assignments such as keyboard input. And we gained insight on how to control LCD panel and clarify some misconception about it, how to program in a case style rather than with if statements.

REFERENCES

Loomis, John. "Digital Labs Using the Altera DE2 Board." *Dr. John S. Loomis*. 18 Nov. 2009. Web. 6 Dec. 2011. <http://www.johnloomis.org/digitallab/>.

Crystalfontz America, Inc. *LCD User Manual*. Web. 1 Dec. 2011. <http://www.cse.yorku.ca/course/3201/CFAH1602BTMCJP-1.pdf>.

"Knowledge Center." *FPGA CPLD and ASIC from Altera*. Altera. Web. 3 Dec. 2011. <http://www.altera.com/>.